|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |
| Институт кибербезопасности и цифровых технологий |
| Кафедра КБ-2 «Прикладные информационные технологии» |

**ПРАКТИЧЕСКАЯ РАБОТА №3 по дисциплине**

**«Технологии и методы программирования»**

***(наименование дисциплины)***

**Тема практической работы   
«Шаблонный метод. Стратегия»**

Студент группы: БИСО-01-20 Дутов А.С.

*(шифр учебной группы) (Фамилия И.О)*

Преподаватель: Лесько С.А.

*(Фамилия И.О)*

Москва, 2023

**Практическая работа №3**

**Стратегия**

class PaymentMethod:

def pay(self, amount):

pass

class CreditCardPayment(PaymentMethod):

def pay(self, amount):

print(f"Payed via Credit Card: {amount}")

class PayPalPayment(PaymentMethod):

def pay(self, amount):

print(f"Payed via PayPal: {amount}")

class Payment:

def \_\_init\_\_(self, payment\_method):

self.payment\_method = payment\_method

def process(self, amount):

self.payment\_method.pay(amount)

# Пример использования

credit\_card\_payment = CreditCardPayment()

payment = Payment(credit\_card\_payment)

payment.process(100)

# Изменение стратегии во время выполнения

pay\_pal\_payment = PayPalPayment()

payment.payment\_method = pay\_pal\_payment

payment.process(50)

Класс PaymentMethod представляет интерфейс, который определяет метод pay(), который должен быть реализован в конкретных реализациях стратегии. В данном примере я создал два класса, CreditCardPayment и PayPalPayment, реализующие этот интерфейс. Класс Payment использует реализации стратегии для проведения платежей. Класс имеет метод process(), который вызывает метод pay() конкретного класса стратегии. В данном примере я создал объекты классов CreditCardPayment и Payment, и использовал их для проведения платежей. Кроме того, я также показал, как можно изменять стратегию во время выполнения, просто изменяя объект, переданный в конструктор объекта Payment.

**Шаблонный метод**

class Process:

def start(self):

self.prepare()

self.execute()

def prepare(self):

print("Preparing the process...")

def execute(self):

pass

class Cook(Process):

def execute(self):

print("Cooking the food...")

class Carpenter(Process):

def execute(self):

print("Building the furniture...")

cooking\_process = Cook()

cooking\_process.start()

carpentry\_process = Carpenter()

carpentry\_process.start()

В этом примере я создал абстрактный класс Process, содержащий методы start(), prepare() и execute(). Метод start() определяет порядок выполнения процесса, а методы prepare() и execute() представляют собой абстрактные шаги процесса, которые должны быть реализованы в подклассах.

Затем я создал два подкласса, Cook и Carpenter, которые определяют собственные реализации метода execute(), в зависимости от своей специфики.

Наконец, я создал объекты классов Cook и Carpenter, вызвал метод start(), чтобы запустить процесс, и получил соответствующие результаты.